PROYECTO FINAL DE CARRERA

TÍTULO DEL PFC: Servidor per gestionar partides multijugador en un joc 2D per Android

TITULACIÓN: Ingeniería Técnica Informática de Sistemas

AUTOR: Edgar José Paz Moreno

DIRECTOR: Lluís Pérez Vidal

FECHA: 18 de Enero de 2011

Índice

1 Introducción

1.1 Oportunidad del proyecto

En la actualidad el mercado que mayor margen de beneficios está obteniendo, es el mundo de los videojuegos. Dentro de éste mercado el sector que más está creciendo es el de los videojuegos on-line[1], ya que el concepto de juego contra una inteligencia artificial empieza a estar algo desgastado.

Hoy en día el mercado cada vez hay más Smartphones[2], por lo tanto es un mercado potencial para sacar rentabilidad a una aplicación, además de que un gran porcentaje de estos dispositivos ya disponen de conexión móvil, y en caso de no tenerla disponen de conexión WIFI[3].

Estas características nos ofrecen la oportunidad de usar este tipo de dispositivos para aprovechar la nueva cuota de negocio, reunificando el concepto de videojuego online para las plataformas móviles, en nuestro caso concreto Android.

La oportunidad es mayor aún en Android, dado que el crecimiento en el mercado de esta plataforma, es exponencial en la actualidad, a lo que hay que sumar el hecho de que sea una comunidad completamente Open Source.

Hoy en día mucha gente se mueve en transporte público, y es una gran momento para poder disfrutar de un juego en el móvil, éste es el principal aliciente del usuario para el consumo de juegos, el hacer los trayectos más amenos.

[1]Que interactúa mediante Internet
[2]Teléfonos que permiten la instalación de programas para incrementar el procesamiento de datos y la conectividad
[3]Conexión de red inalámbrica

1.2 Objetivo

El objetivo de este proyecto es conseguir desarrollar una conexión en que dos jugadores consigan conectarse de manera que puedan jugar entre ellos sin estar físicamente al lado uno del otro.

El objetivo principal de este juego, puede dividirse en varios objetivos secundarios, todos y cada uno de los cuales deben ser asumidos para llevar a cabo el proyecto.

Un primer objetivo sería el desarrollo, o adaptación, de un servidor de contenidos que nos permita gestionar toda la información relacionada con los jugadores que deseen establecer una partida online desde sus dispositivos.

En segundo lugar es necesario llevar a cabo la implementación de una extensión que se encargue de manejar las características individuales del juego que en este proyecto se plantean.

Por último es necesario llevar a cabo la adaptación de la aplicación android desarrollada para juego en local, para poder manejar todos los eventos derivados de la comunicación de dos jugadores online, manteniendo una experiencia de usuario y juego agradable en todo momento.

1.3 Estructura del documento

Este documento consta de 9 apartados, los cuales detallaremos a continuación:

El primer apartado consta de una breve introducción acerca de la oportunidad que se presenta con este proyecto, y los objetivos que se quieren alcanzar con dicho proyecto

El segundo apartado explica la tecnología utilizada para poder implementar el servidor, en este apartado encontraremos todo lo referente al servidor Smartfox.

En el tercero explicaremos el funcionamiento y características de la aplicación desarrollada en Android, *Lamb Pastor*.

El cuarto apartado consta de la explicación de la parte multijugador de *Lamb Pastor.*

El quinto apartado analizaremos el coste que nos supone la infraestructura del servidor y todos los servicios que deben ir asociados a él para que el funcionamiento del servidor sea el correcto.

El sexto apartado profundizará acerca de las dificultades encontradas a la hora de implementar el servidor

El séptimo apartado contendrá las conclusiones acerca del proyecto.

El octavo incluirá toda la bibliografía usada en el proyecto.

Y por último el noveno contendrá los anexos.

2 SmartFox Server

2.1 Definición

SmartFox Server es un servidor desarrollado en Java[4] que permite implementación de servidores de videojuegos mediante la implementación de extensiones, las cuales serán las encargadas de gestionar la información que se enviará entre el servidor y los clientes.

Además de estar desarrollado en *Java*, se despliega sobre un servidor *Tomcat*[5] para poder acceder vía web al apartado de administración vía web.

Para realizar la conexión entre el servidor y el cliente se realiza mediante Json[6] para poder realizar la conexión entre el cliente y servidor. Protocolo es comúnmente usado en las conexiones entre servidor y cliente ya que es un sistema sencillo y muy ligero.

Este servidor ha sido desarrollado para poder interactuar con diferentes plataformas lo que hace muy versátil este servidor de manera que lo hace que la parte de programación se simplifique mucho, y sea más cómodo de cara al programador.

[4]Lenguaje de programación originalmente desarrollado por Sun Microsystems, adquirida por Oracle, para aplicaciones software independiente de la plataforma.
[5]Servidor web que además puede contener aplicaciones.
[6] Formato ligero para el intercambio de datos.

2.2 Características

– Multiplataforma: al estar desarrollado sobre Java puede correr sobre cualquier *Sistema Operativo*[7].

– Configuración visual y administración: Toda la configuración se realiza a través de la nueva AdminTool que también proporciona funciones avanzadas de estadísticas en tiempo real, gestión de tiempo de ejecución de la zona / habitación / usuario, la gestión de Ban y mucho más.

La nueva AdminTool también ofrece una arquitectura modular que permite la integración de terceros plug-ins.

– Alto rendimiento de red:

  – Alta escalabilidad con diseño no bloqueante, que permite estabilidad y rendimiento.
  – Arquitectura con mejor desempeño en el consumo de memoria.
  – Baja latencia y reduce al mínimo copia innecesaria de búfer[8] / memoria.
  – Soporte centralizado para personajes no jugadores a nivel de sesión.
  – Gestión activa de desconexión.
  – Modelo de subprocesos configurables.
  – *HTTP*[9] de alto rendimiento de túnel para los clientes no pueden establecer conexiones de *socket*[10].
  – Sistema de reconexión transparente: permite que las sesiones de reconexión se realice de forma transparente a la aplicación sin perder su estado después de la desconexión.

[7]Es el programa o conjunto de programas que efectúan la gestión de los procesos básicos de un sistema informático, y permite la normal ejecución del resto de las operaciones.
[8]Espacio de memoria, en el que se almacenan datos para evitar que el programa o recurso que los requiere, ya sea hardware o software, se quede sin datos durante una transferencia.
[9] Protocolo usado en cada transacción de la World Wide Web.
[10]Concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada.

- Protocolo altamente efectivo: el protocolo binario ofrece importantes mejoras en el rendimiento del servidor y la red, entregando un promedio de 5 veces en la codificación de los tiempos y 6 veces en la reducción del tamaño del paquete.

- Seguridad:
  - Los datos de acceso siempre se transmiten a través de mecanismo de seguridad para evitar la detección de contraseñas.
  - El Administrador de autorización deja crear cualquier número de perfiles de usuario que permite a los diferentes clientes acceder sólo a una parte de las características del servidor de acuerdo a sus propios permisos.
  - Filtro que protege el servidor de los ataques de las inundaciones y ofrece una configuración de grano fino para cada solicitud de servidor.
  - El mejor filtro de palabras, soporta expresiones regulares y los modos de *white-list/*[11]*black-list*[12], ofreciendo una herramienta más flexible para insultos. El filtro se puede aplicar a los mensajes públicos / privados, de habitaciones y de nombres de usuario y acceder a través de API simple de código del lado del servidor. También ofrece eventos para el registro y más personalizaciones.
  - Sistema Avanzado de *Banneo*[13]: proporciona herramientas para realizar de manera manual y automática de la prohibición, la persistencia y la configuración para cada usuario baneado.
  - Filtrado *IP*[14] permite controlar el número máximo de tomas de corriente procedente de la misma dirección IP.

[11]Lista o registro de entidades que, por una razón u otra, pueden obtener algún privilegio particular, servicio, movilidad, acceso o reconocimiento.
[12]Lista de personas, instituciones u objetos que deben ser discriminados en alguna forma con respecto a los que no están en la lista.
[13]Restricción ya sea total, parcial, temporal o permanente, de un usuario dentro de un sistema informático, generalmente una red.
[14]Etiqueta numérica que identifica, de manera lógica y jerárquica, a un interfaz de un dispositivo dentro de una red que utilice el protocolo IP (Internet Protocol), que corresponde al nivel de red del protocolo TCP/IP.

2.3 Tecnologías que usa

2.3.1 Tomcat

Funciona como un contenedor de servlets[15] desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de *JavaServer Pages (JSP)*[16] de *Sun Microsystems*[17].

Tomcat no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

Tomcat puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones.

Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

[15]Programas que se ejecutan en un servidor mediante interacción web.
[16]Tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.
[17]Fue una empresa informática recientemente (2009) adquirida por Oracle Corporation, anteriormente parte de Silicon Valley, fabricante de semiconductores y software.

- Estructura de directorios:
  - bin - arranque, cierre, y otros scripts y ejecutables.
  - common - clases comunes que pueden utilizar Catalina y las aplicaciones web.
  - conf - ficheros XML y los correspondientes DTD para la configuración de Tomcat.
  - logs - logs de *Catalina*[18] y de las aplicaciones.
  - server - clases utilizadas solamente por Catalina.
  - shared - clases compartidas por todas las aplicaciones web.
  - webapps - directorio que contiene las aplicaciones web.
  - work - almacenamiento temporal de ficheros y directorios.

[18]Catalina es el nombre del contenedor de servlets del Jakarta Tomcat.

2.3.2 Json

La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a *XML*[19] en *AJAX*[20]. Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON.

En *JavaScript*[21], un texto JSON se puede analizar fácilmente usando el procedimiento eval (), lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

En la práctica, los argumentos a favor de la facilidad de desarrollo de analizadores o del rendimiento de los mismos son poco relevantes, debido a las cuestiones de seguridad que plantea el uso de eval () y el auge del procesamiento nativo de XML incorporado en los navegadores modernos.

Por esa razón, JSON se emplea habitualmente en entornos donde el tamaño del flujo de datos entre cliente y servidor es de vital importancia (de aquí su uso por Yahoo, Google, etc., que atienden a millones de usuarios) cuando la fuente de datos es explícitamente de fiar y donde no es importante el no disponer de procesamiento XSLT para manipular los datos en el cliente.

Si bien es frecuente ver JSON posicionado contra XML, también es frecuente el uso de JSON y XML en la misma aplicación. Por ejemplo, una aplicación de cliente que integra datos de Google Maps con datos meteorológicos en *SOAP*[22] hacen necesario soportar ambos formatos.

[19]Metalenguaje extensible de etiquetas.
[20]Técnica de desarrollo web para crear aplicaciones interactivas.
[21]Lenguaje de programación interpretado.
[22]Protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

Cada vez hay más soporte de JSON mediante el uso de paquetes escritos por terceras partes. La lista de lenguajes soportados incluye, *ActionScript, C, C++, C#, ColdFusion, Common Lisp, Delphi, E, Eiffel, Java, JavaScript, ML, Objective CAML, Perl, PHP, Python, Rebol, Ruby, y Lua*[23]. En diciembre de 2005 Yahoo![24] comenzó a dar soporte opcional de JSON en algunos de sus servicios web.

El término JSON está altamente difundido en los medios de programación, sin embargo, es un término mal descrito ya que en realidad es solo una parte de la definición del estándar ECMA-262 en que está basado Javascript. De ahí que ni Yahoo, ni Google emplean JSON, sino LJS.

Una de las cualidades intrínsecas de Javascript denominada LJS (Literal Javascript) facilita el flujo de datos e incluso de funciones, para la cual no requiere la función eval () si son datos los que se transfieren como en el caso de XML.

Todo lo referente a transferencia de datos en todos sus tipos, incluyendo arrays, booleans, integers, etc. no requieren de la función eval (), y es precisamente en eso en donde supera por mucho JavaScript al XML, si se utiliza el LJS y no la incorrecta definición de JSON.

[23]Lenguajes de programación.
[24]Empresa global de medios con sede en Estados Unidos, cuya misión es "ser el servicio global de Internet más esencial para consumidores y negocios"

2.3.3. COMPARACIÓN CON XML Y OTROS LENGUAJES DE MARCADO

XML goza de mayor soporte y ofrece muchas más herramientas de desarrollo (tanto en el lado del cliente como en el lado del servidor). Hay muchos analizadores JSON en el lado del servidor, existiendo al menos un analizador para la mayoría de los entornos. En algunos lenguajes, como Java o PHP, hay diferentes implementaciones donde escoger. En JavaScript, el análisis es posible de manera nativa con la función eval (). Ambos formatos carecen de un mecanismo para representar grandes objetos binarios.

Con independencia de la comparación con XML, JSON puede ser muy compacto y eficiente si se usa de manera efectiva. Por ejemplo, la aplicación *DHTML*[25] de búsqueda en BarracudaDrive recibe los listados de directorio como JSON desde el servidor. Esta aplicación de búsqueda está permanentemente consultando al servidor por nuevos directorios, y es notablemente rápida, incluso sobre una conexión lenta.

Los entornos en el servidor normalmente requieren que se incorpore una función u objeto analizador de JSON. Algunos programadores, especialmente los familiarizados con el lenguaje C, encuentran JSON más natural que XML, pero otros desarrolladores encuentran su escueta notación algo confusa, especialmente cuando se trata de datos fuertemente jerarquizados o anidados muy profundamente.

YAML[26] es un superconjunto de JSON que trata de superar algunas de las limitaciones de éste. Aunque es significativamente más complejo, aún puede considerarse como ligero. El lenguaje de programación Ruby utiliza YAML como el formato de serialización por defecto. Así pues, es posible manejar JSON con bastante sencillez.

[25] Designa el conjunto de técnicas que permiten crear sitios web interactivos utilizando una combinación de lenguaje HTML estático, un lenguaje interpretado en el lado del cliente (como JavaScript), el lenguaje de hojas de estilo en cascada (CSS) y la jerarquía de objetos de un DOM.

[26] Formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python, Perl.

2.4 Árbol de directorios

A continuación definiremos la información que encontramos dentro del directorio del proyecto.

- Carpeta *bin:* en un principio no contiene nada, pero más adelante explicaremos como le podemos dar uso.
- Carpeta *lib*: contiene todas las librerías usadas por la extensión del servidor.
- Carpeta *src*: este directorio contiene todo el código fuente de la extensión del servidor.

2.5 Clases en la extensión del servidor

Es el momento de detallar una a una las diferentes clases que componen la extensión del servidor.

GameState

Esta clase contiene una codificación para poder determinar si el juego se encuentra en marcha, ha acabado en empate o ha acabado con algún vencedor.

LambPastorExtension

Esta es la clase principal, es la instancia que se lanza cuando un cliente conecta con el servidor usando la extensión de LambPastor. La clase contiene las siguientes variables:

- LambGameBoard: más adelante trataremos esta clase.
- GameStarted: booleano que nos indicará si el juego está iniciado o no.
- LastGameEndResponse: más adelante trataremos esta clase.
- dentro: contiene el número de ovejas que ya han sido introducidas en el cerco.
- ovejasDentro: vector de booleanos que contendrá si una oveja ha sido metida en el cerco por el jugador 1(booleano igual a cierto) o por el jugador 2(booleano igual a falso).
- Version: es una cadena de caracteres que contiene la versión de la extensión.

- MoveControl: clase que trataremos más adelante.

-init ()

Método que se llama cuando se crea una sala invocando a esta extensión del servidor. Inicializa todas las variables del servidor, pone *dentro* a 0, crea los objetos LambGameBoard, LastGameEndResponse, ovejasDentro y MoveControl.

A continuación registramos los manejadores para que el servidor esté a la escucha de las peticiones de los clientes. Los manejadores registrados corresponderán al comando "*move"* que va asociado a la clase MoveHandler que será el encargado de gestionar y actualizar los movimientos de los elementos del juego (perros y ovejas).

El comando "*restart"* va asociado a la clase RestartHandler encargada de gestionar el juego cuando se reinicia la partida, y por último al comando "*ready"* que es el encargado de recibir la confirmación por parte de los jugadores de que están listos y avisarlos cuando esto suceda para que el juego de comienzo.

Por último en este método se registran manejadores de evento y se inicializa la variable *gameStarted* a falso. Estos manejadores se ocuparán de atender los eventos: usuario desconectado que es gestionado por la clase *OnUserGoneHandler*, usuario abandona la sala también gestionada por la clase *OnUserGoneHandler* y el último que administrará los usuarios que son espectadores que la gestiona la clase *OnSpectatorToPlayerHandler*.

-destroy ()

Se encarga de destruir el objeto *LambPastorExtension* creado en memoria, además deja constancia en el log del servidor que esto ha pasado.

-getGameBoard ()

Devuelve el objeto privado de la clase *GameBoard.*

-getdentro ()

Devuelve el número de ovejas que ya han sido metidas dentro del cerco.

-increaseDentro ()

Incrementa el valor de la variable *dentro* en 1.

-isGameStarted ()

Devuelve cierto si la partida ha sido iniciada y falso en caso contrario.

-startGame ()

Este método se lanza cuando ambos jugadores están listos para empezar el juego. *GameStarted* pasa a tener valor cierto, el servidor recupera los usuarios implicados en la partida, para poder diferenciar entre el jugador 1 y el jugador 2.

Se crea el objeto *resObj* que contiene varias *tablas de hash#* para poder comunicarse con los usuario y poder intercambiar información del juego. El servidor dentro de esta variable incluye los nombres de cada usuario para que cada uno de ellos sepan que jugador son (jugador 1 o jugador 2).

Acto seguido se inicia la variable *MoveControl* que más adelante detallaremos en que consiste, pero que en este momento inicializa las posiciones de las ovejas y de los perros.

Una vez inicializado todo y añadido a la variable *resObj* y se manda a los usuarios mediante el método *send* y con el parámetro "*start"*.

-stopGame ()

Este método se encarga de parar el juego.

-getGameRoom ()

Devuelve el objeto *Room* (sala en la cual se encuentran los jugadores actualmente).

-getLastGameEndResponse ()

Devuelve el objeto *LastGameEndResponse*.

-setLastGameEndResponse (LastGameEndResponse)

Establece la variable privada con el valor que se pasa por parámetro.

-updateSpectator (User user)

Pasa al usuario que se pasa por parámetro a estado de espectador.

- setovejasDentro (int i, boolean player1)

Introduce en la posición i del vector *ovejasDentro* el valor booleano *player1*.

- getovejasDentro ()

Devuelve el vector de *ovejasDentro*.

LambGameboard

Contiene el estatus del juego, es decir controla si el juego ha finalizado o aún debe seguir (comprueba si todas las ovejas están ya en el cerco.

LastGameEndResponse

Esta clase se encarga de guardar el último comando que han enviado los usuarios al servidor.

MoveControl

En esta clase haremos especial hincapié ya que es la que contiene la inicialización de los componentes del juego y se encarga de actualizar la posición de las ovejas.

- Variables

- ovejas: vector de *Animal*, que contiene los datos referentes a las ovejas.
- perro1: objeto que contiene la información del perro controlado por el jugador 1.
- perro2: objeto que contiene la información del perro controlado por el jugador 2.
- fin: booleano que adquirirá el valor cierto cuando la extensión reciba la orden de terminar el juegos.
- collides: booleano que se pondrá a cierto cuando un perro colisione con una oveja.

- MoveControl ()

En este método se inicializa el vector de ovejas, se recorre dicho vector mientras se van creando e inicializando todos los valores de las ovejas. Una vez inicializadas las ovejas se procede con la inicialización de los perros. Por último se inicializa *fin* a false y *collides* también a falso.

- run ()

Método que extiende de la clase *Thread*[27], cuando es invocado éste hilo corre en paralelo del hilo principal. Mientras el juego no termine el bucles seguirá activo en el momento que uno de los jugadores modifique el camino que siguen las ovejas, la variable collides se pondrá a cierto, y se recalculará un nuevo camino para dicha oveja.

[27]Hilo de ejecución.

Una vez hecho esto volveremos a poner *collides* a falso para que pueda atenderse otra colisión.

- Fin ()

Pone la variable *fin* a cierto, y por lo tanto mata ya al thread que está a la espera de las colisiones.

Clase Animal

Para poder almacenar la información de los perros y las ovejas en la clase *MoveControl* se decidió incluir una clase genérica que englobara elemento en común de estos dos elementos.

Estos parámetros son: posición en el eje de las x, posición en el eje de las y, un *path*[28] en el eje de las x, *path* en el eje de las y, y por ultimo una variable *path.*

MoveHandler

Esta clase es la encargada de atender las peticiones de los clientes cuando mandan una petición de que un perro se ha movido. Las variables que contiene son meramente etiquetas para facilitar la programación de la clase.

- handleClientRequest (User user, ISFSObject params)

El parámetro *user* que le llega a la clase nos indica que usuario ha sido el que ha enviado la petición. Por otro lado el parámetro *params* contiene toda la información que le ha enviado el usuario al servidor.

Primero comprobamos si el juego se ha iniciado, en caso afirmativo, cuando un usuario envía la posición de su perro, primero identificamos si se trata del perro controlado por el jugador1 o del perro controlado por el jugador2.

[28]Del inglés, Camino.

En el caso que no sean ninguno de los anteriores pasamos a comprobar si el dato enviado ha sido el cambio de posición de una oveja, si es así se la enviamos al otro jugador para que lo actualice en la interfaz.

Por último si no es ninguno de los anteriores comprobamos si el parámetro enviado ha sido *"fin"* en este caso le diremos a la extensión que la partida ha finalizado y que por tanto avise a los jugadores con el resultado final de la partida.

- checkBoardState (LambExtension gameExt)

Este método comprueba el estado del juego, si ha finalizado. En caso negativo no hace nada.

onSpectorToPlayerHandler

Esta clase se ocupa de mantener a los usuarios en modo espectador hasta que haya el número de usuarios indicado para poder iniciar el juego.

onUserGoneHandler

Clase que se encarga de detectar cuando un usuario abandona la sala en la que se encuentra actualmente, o cuando el usuario se desconecta del servidor.

ReadyHandler

Clase que se encarga de recibir las peticiones de los usuarios conforme ya están listos para iniciar el juego. En el momento que la clase detecta que hay dos usuarios listos para iniciar la partida en la misma sala, manda la orden a la extensión conforme ya puede iniciar la partida.

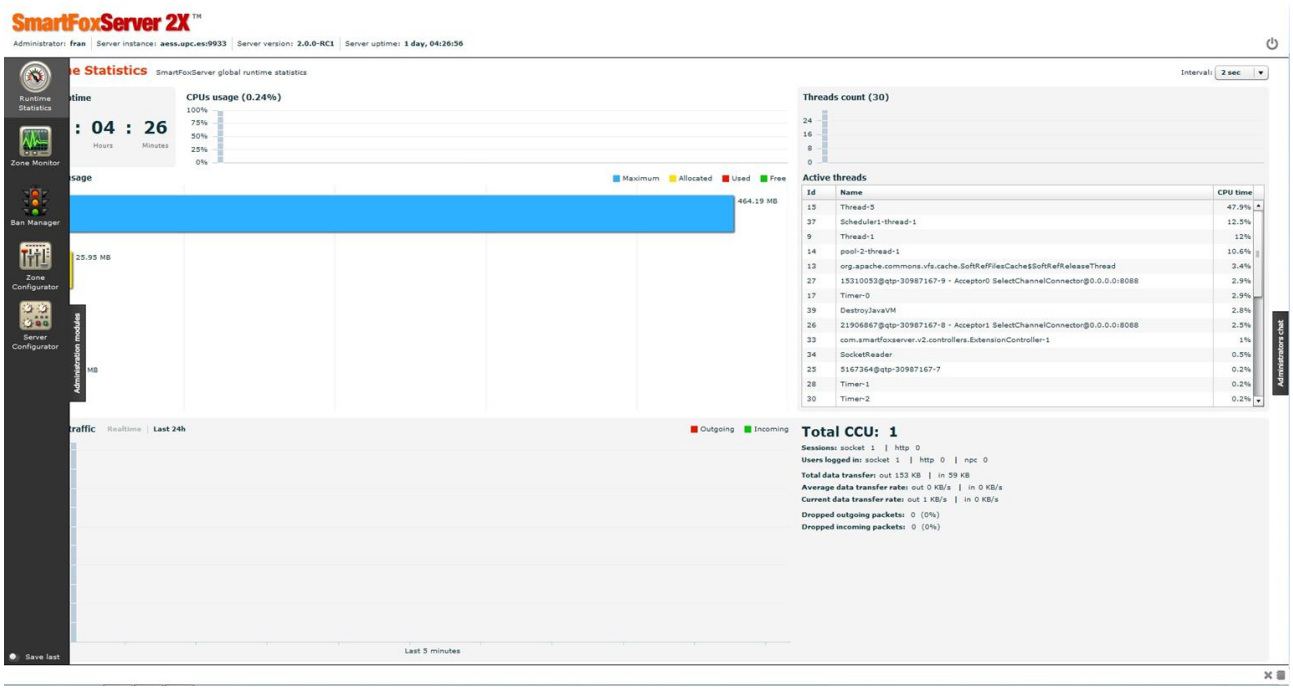
RestartHandler

Esta clase se encarga de reiniciar el juego en caso de recibir dicha petición.

2.6 Zona de administración

En este apartado hablaremos sobre la zona de administración que publica el servidor para poder configurar las zonas[29] y las salas. Esta zona nos permite manejar todo aquello que tiene que ver con nuestro juego, para poder ofrecer al usuario la mejor experiencia de juego posible, en todo momento.

Al iniciar sesión en la parte de administración nos encontramos con la siguiente pantalla:



En esta pantalla podemos observar la *zona de monitorización*, desde aquí comprobamos el estado de los recursos del servidor: Uso de la CPU, uso de Memoria RAM, el tiempo que hace que el servidor está activo y consumo del tráfico de red entrante y saliente.

Todos estos parámetros que nos ofrece esta pantalla, nos ayudan a saber las dimensiones que debe tener el servidor en el que debemos alojar el servidor del juego.

[29]Hace referencia a cada una de las extensiones que tiene el servidor.

Es importante ajustar de forma correcta estos valores, ya que en caso de que los recursos que demande el servidor, sean superiores a los que ofrece el hardware en el que esta alojado, provocará problemas en la comunicación entre dispositivos móviles, haciendo que nuestro objetivo primordial que es la experiencia de juego agradable, se vea claramente afectada.

En la parte derecha encontramos un chat de administradores, una opción muy cómoda si hay más de un administrador en la aplicación. Este chat nos permite mantener una comunicación entre administradores, para poder solventar posibles problemas o incidencias que hayan detectado.

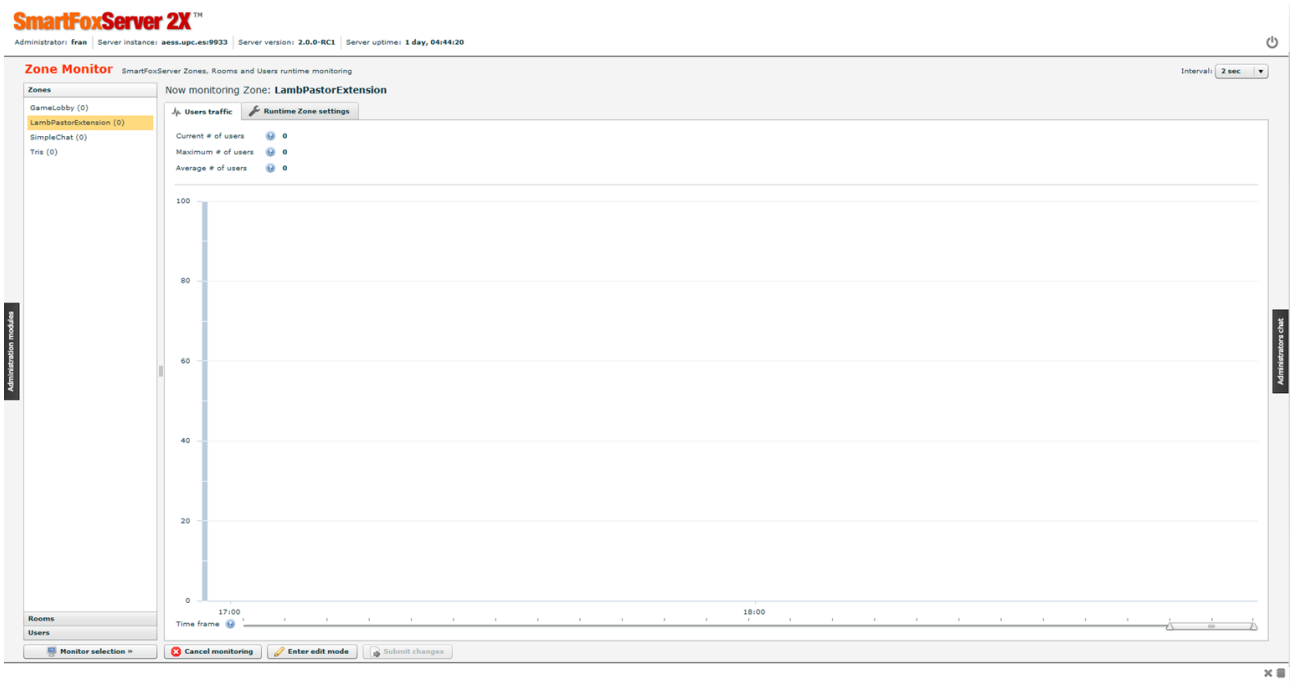
Esto nos permite reducir costes de mantenimiento del servidor, ya que no es necesario mantener reuniones, con sus elevados costes, ya que los administradores pueden mantenerse en contacto, gracias al propio servidor en todo momento.

En la parte izquierda de la pantalla vemos las diferentes pestañas a las que podemos acceder. Entre ellas encontramos *"Runtime Statistics"*, *"Zone Monitor"*, *"Ban manager"*, *"Zone configurator"*, y por último *"Server Configuration"*.

Estas cuatro zonas mencionadas, nos ayudan a agrupar más concretamente, y reduciendo el tiempo de búsqueda entre la opción deseada, para poder realizar las operaciones de mantenimiento más apropiadas en cada momento, y haciendo más intuitiva la interfaz de usuario del propio servidor.

El primero (*"Zone Monitor"*) es el que ya hemos comentado, así que empezaremos a analizar el resto de pestañas.

En *"Zone Monitor"* podemos monitorizar el tráfico que se genera en cada una de las zonas, es decir monitoriza la afluencia de usuarios en las distintas extensiones. Esta monitorización también se puede llevar a cabo sobre cada una de las salas creadas dentro de las zonas. A continuación mostramos una captura:



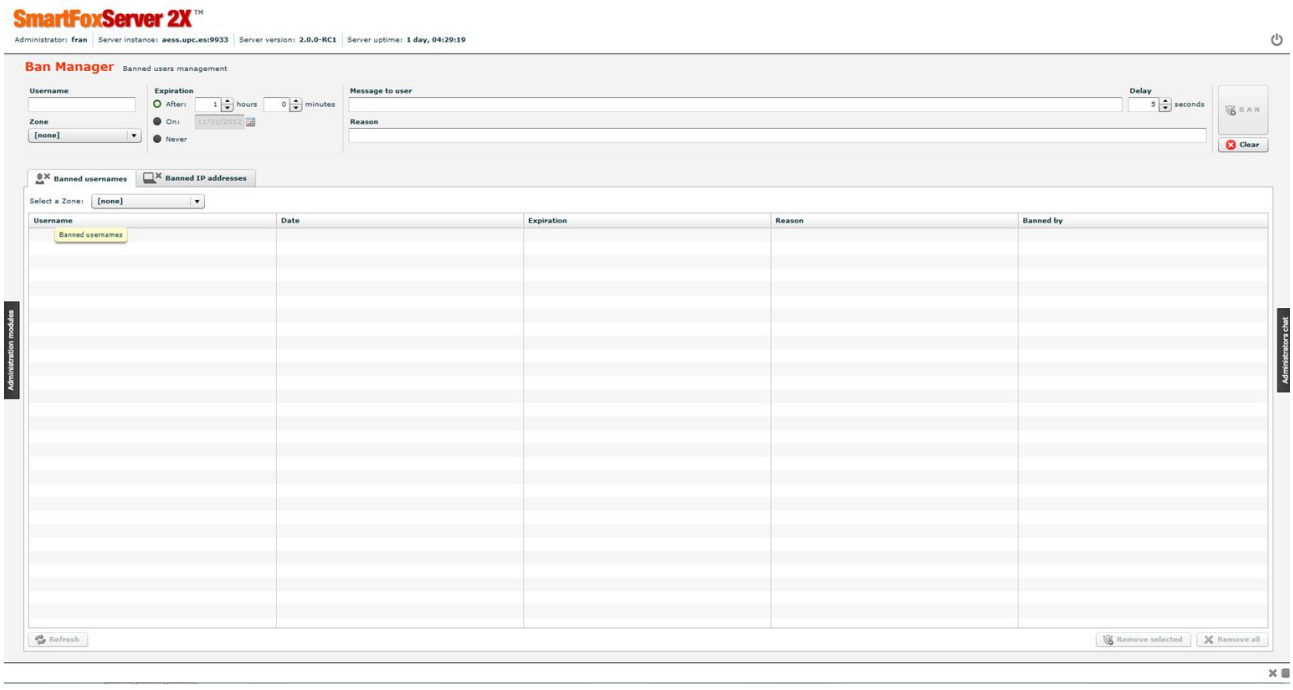
Esta parte del servidor, nos permite saber el rendimiento que tiene cada una de las diferentes extensiones de las que se puede componer el servidor, permitiendo saber que juego de los diferentes que puede alojar, es el que nos esta reportando un mayor beneficio, o en el caso concreto, cual esta teniendo problemas de rendimiento.

Podemos ver gráficos de tráfico entrante y saliente de la extensión, o lo que es lo mismo, de un juego concreto, lo que nos puede ayudar a redimensionar el hardware del servidor ante un crecimiento de la demanda, o a calcular el número de usuarios que usar nuestra aplicación, y por lo tanto los beneficios que nos puede reportar en un futuro.

En *"Ban manager"* encontramos una interfaz completa para poder bannear usuarios. Además de bannear usuarios que no cumplan las normas, podemos definir una serie de reglas para echar a gente de una sala incluso del servidor.

Usaremos esta parte del servidor como parte vital de nuestro objetivo de ofrecer una experiencia de juego lo más agradable posible en todo momento. El uso que la daremos es mantener bajo control en todo momento a todos los usuarios que interactúen con nuestro servidor, evitando disputas entre ellos, que lleven a otros usuarios a abandonar el uso de la aplicación.

Estas reglas suelen ser insultos, palabras malsonantes o cualquier cosa que pueda interpretar el administrador como ofensivas. Además SmartFox permite el uso de listas negras, por lo que podemos prohibir el acceso a un usuario en concreto, incluso a una IP. Una muestra de la interfaz sería la siguiente:



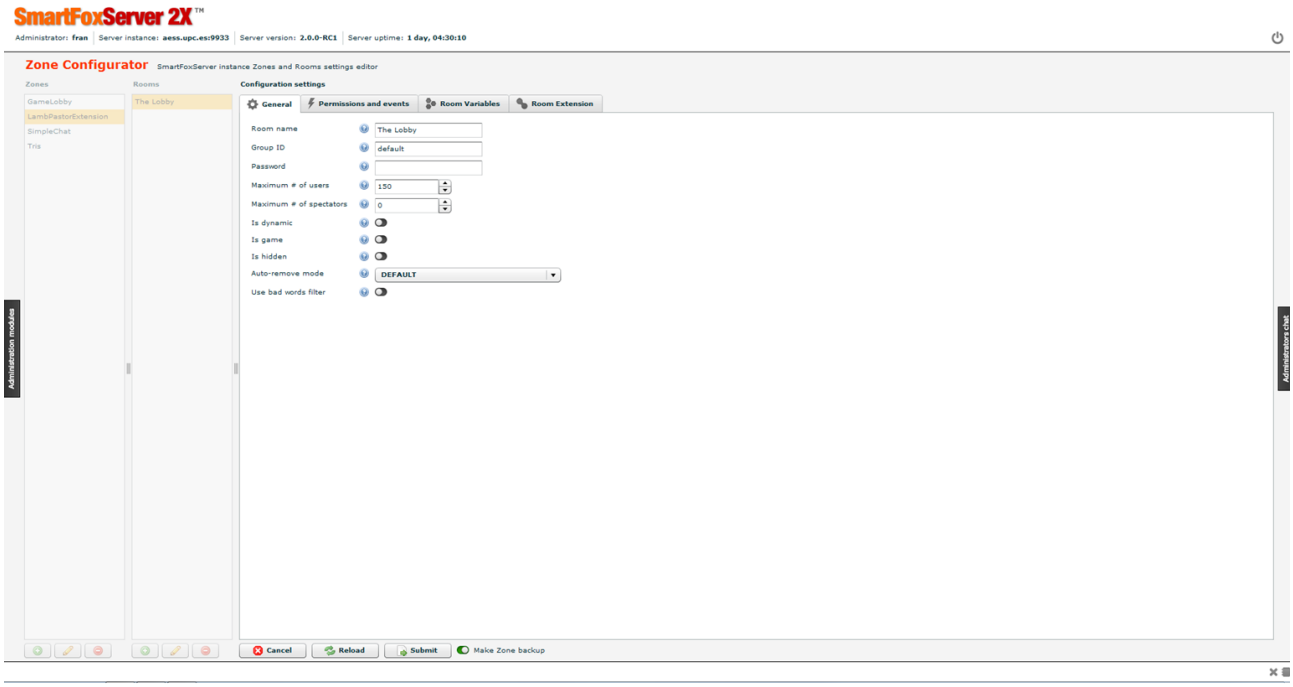
Esta zona tiene un potente motor de gestión de usuarios, que nos permite asegurarnos de que un mismo usuario no podrá usar diferentes nombres para poder perjudicar la experiencia de juego, crear grupos de pruebas reducidos a usuarios o terminales concretos, etc.

En la pestaña de *"Zone configurator"* podemos configurar todos los elementos que tienen que ver con nuestra zona. Dentro de la zona podemos configurar diferentes aspectos. En la primera pestaña de este apartado podremos configurar los parámetros más generales como el nombre de la zona, número máximo de usuarios, número máximo de espectadores.

Usando los parámetros de configuración que nos ofrece esta nueva parte, podemos controlar que el tráfico máximo que soporta cada zona, no sea excesivo y por lo tanto no existan retrasos en el juego, que puedan incomodar a los usuarios de la aplicación.

También nos permite configurar el tiempo máximo que un jugador puede esperar a iniciar una partida, evitando que se creen muchas partidas simultáneamente, y no llegue a jugarse ninguna, o que se haga un uso indebido del servidor.

Además de todos los detalles comentados, podemos gestionar el número máximo y mínimo de rooms que puede gestionar cada zona, incluso podemos definir un password que restringa el acceso a un grupo de usuarios controlado, con el objetivo de optimizar el rendimiento del hardware.



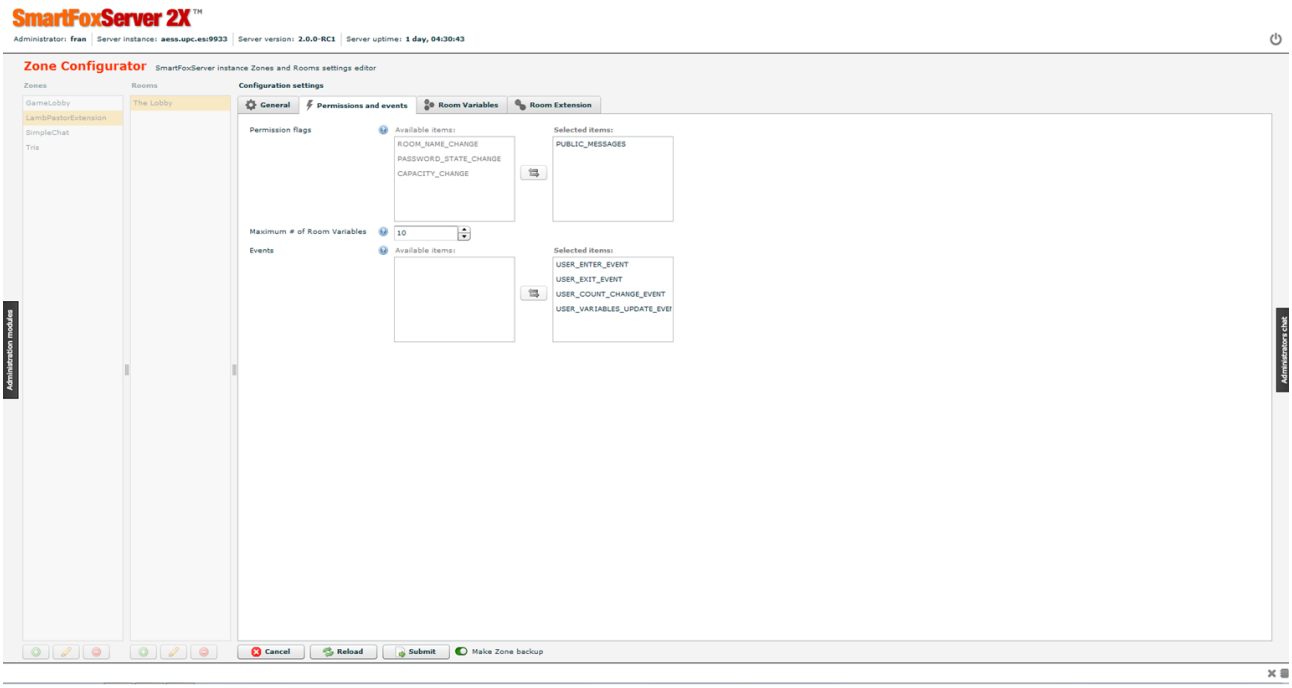
En la siguiente pestaña configuraremos los diferentes permisos de la zona y los eventos que tiene asociada. Estos permisos pueden ser cambio de alias, cambio de nombre de sala, publicar mensajes, etc.

Mediante la configuración que nos ofrece esta nueva pestaña, podemos manejar todos y cada uno los eventos y permisos que se derivan de la interacción de los usuarios con las salas, lo que nos permite saber en todo momento lo que sucede dentro de esta, permitiendo asumir nuestro objetivo de una experiencia de juego agradable.

Entre los eventos que podemos manejar están la entrada o salida de usuarios en la zona, la actualización de variables o configuraciones relacionadas con esta, o una serie de eventos personalizados que podemos definir mediante ficheros XML personalizados.

Con el control de todos estos eventos podemos gestionar y redimensionar el número de juegos que podemos mantener en ejecución en cada momento dentro del servidor que tenemos desplegado.

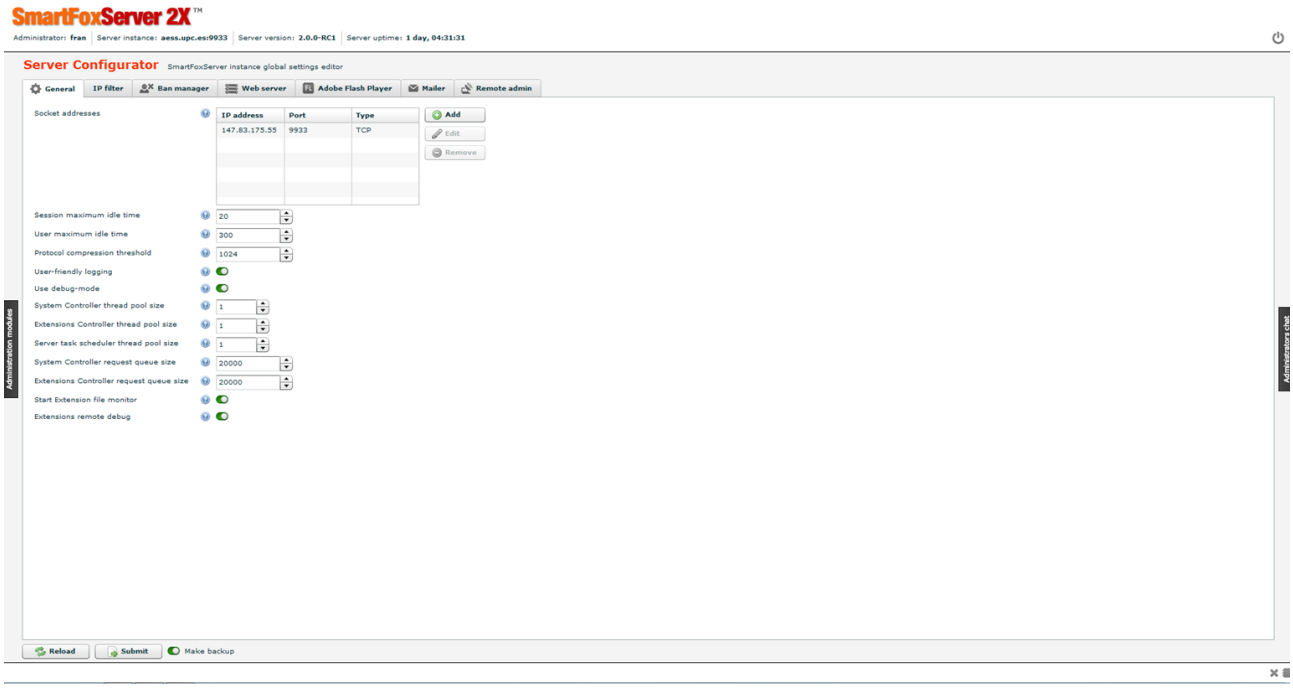
La interfaz es la siguiente:



Por último tenemos la pestaña *"Server Configuration"* en la que encontramos toda la configuración referente al servidor. Como en casos anteriores disponemos de diferentes pestañas para configurar diferentes partes del servidor.

La primera de ellas es la pestaña *General* en la que encontramos un apartado para añadir más servidores a la lista de direcciones IP, además podemos configurar el tiempo que un usuario puede estar conectado sin enviar ningún dato al servidor.

En esta pestaña podemos configurar el servidor de cara al desarrollador, nos permite activar el modo debug. Este modo genera una serie de mensajes y logs que posteriormente podemos analizar para detectar fallos en el servidor, o para optimizar su configuración para obtener un mayor rendimiento del sistema, y por lo tanto un juego más fluido de cara al usuario final.

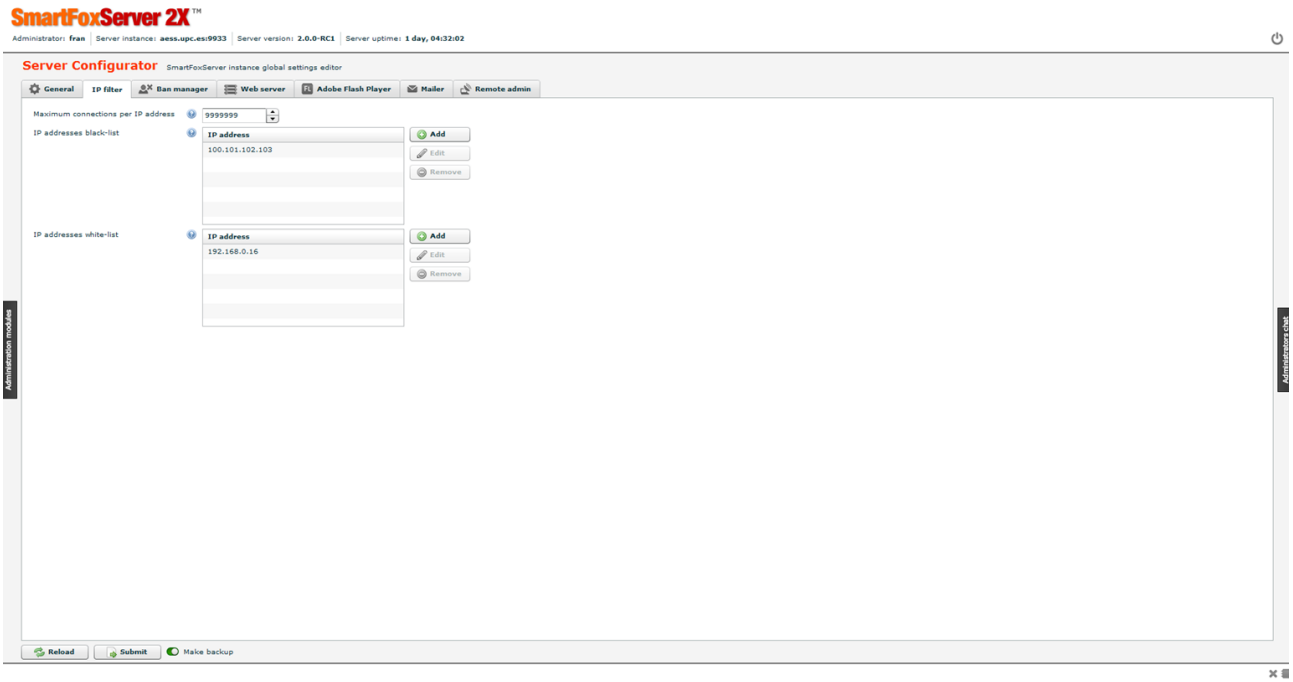


El siguiente apartado de la zona de configuración de servidor es la de filtrado IP, como hemos comentado anteriormente aquí podemos configurar la *lista negra* y la *lista blanca.*

Mediante estas dos listas podemos añadir un extra de seguridad a nuestro servidor, ya que podemos o bien definir una serie de usuarios que no pueden acceder al servidor, por una serie de motivos que los administradores han considerado válidos, siendo este el caso de la lista negra, o BlackList.

La segunda lista, WhiteList, nos permite ser mucho más restrictivos aún, ya que únicamente podrán acceder al servidor aquellos dispositivos cuya IP se encuentre en la lista, la utilidad de esta lista puede ser probar nuevas extensiones o funcionalidades del servidor, de forma que solo permitiremos acceder a una serie de usuarios.

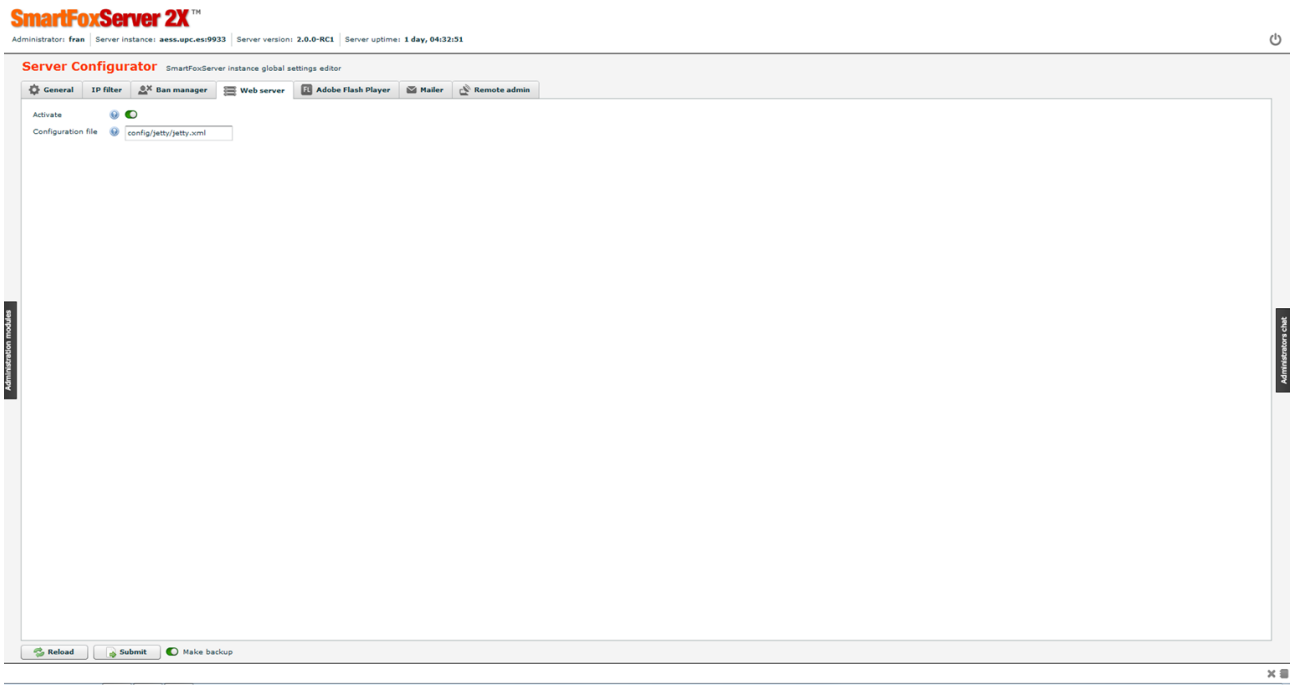
Estos usuarios serán elegidos por los administradores del servidor, bien por que han aceptado probar una parte en desarrollo del sistema, asumiendo los posible fallos e interrupciones que pueden generarse por estar en fase de pruebas, o bien por ser usuarios que puedan reportar información importante a estos, por su experiencia en el mundo de los juegos online, que ayuden a mejorar.



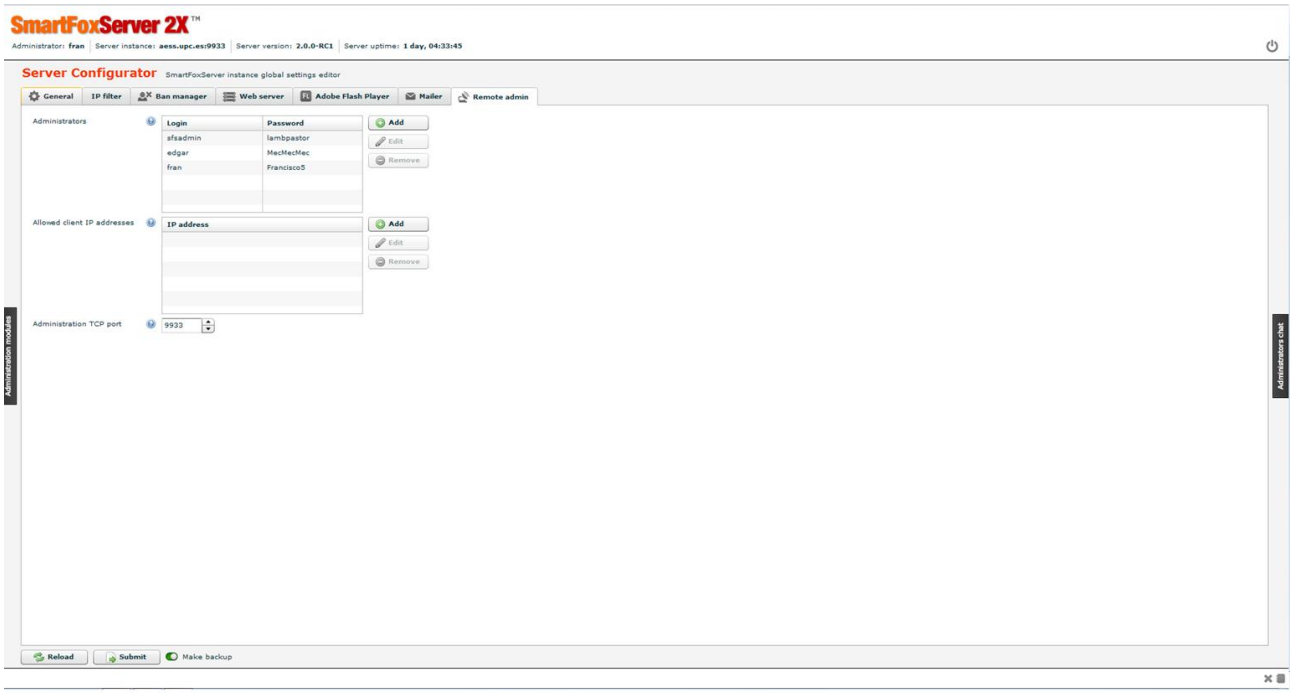
El próximo apartado es el *Web Server*. En este apartado sólo definimos de qué archivo debe coger el servidor los datos de configuración del servidor web.

En este archivo definiremos sobre que contenedor de aplicaciones dentro de nuestro servidor, en este caso Tomcat, debe desplegarse el servidor de juegos, además del contenedor de aplicaciones, también definiremos el puerto público sobre el que se iniciará para evitar colisiones con otros servicios ofrecidos por el misma máquina.

Este fichero es sumamente importante, ya que el primer inicio del servidor depende de su correcta definición, todas la variables básicas del sistema deben definirse correctamente, una vez iniciado el servidor por primera vez, todas la modificaciones que hemos comentado hasta el momento sobre la configuración del servidor realizadas a través de la interfaz web, se verán reflejadas en este fichero.



Para finalizar con este apartado tenemos la pestaña de *Remote Login* en la que podremos configurar diferentes administradores, así como definir ip's desde las cuales serán las únicas desde donde se podrá acceder a la parte de administrador del servidor.



3 Lamb Pastor

En este proyecto hemos decidido acondicionar el servidor para crear opción de partida On-line multijugador para éste juego.

3.1 En que consiste

El juego está basado en las competiciones de *Gos d'atura*[30], donde un perro debe meter a un grupo de ovejas en un cerco. El perro es controlado por el jugador mediante una pulsación encima del perro y arrastrándolo por la pantalla. El juego finaliza cuando se acaba el tiempo (que es de 5 minutos), o el jugador consigue meter a todas las ovejas dentro del cerco.

El propio motor del juego es el encargado de simular un movimiento de las ovejas, de manera que estas huyan usando toda la amplitud del escenario, complicando, siempre dentro un límite razonable, la tarea del usuario de llevar estas dentro del corral.

Cuando se inicia el juego, se inicia automáticamente la cuenta atrás del tiempo máximo que el usuario tiene para conducir a las ovejas con su perro al interior del cercado. El objetivo es conducir a todo el rebaño al interior de este antes de que el tiempo se agote, resultando así cumplido el objetivo del juego y por lo tanto ganando la partida.

En este momento es cuando el usuario debe pulsar sobre el perro y empezar a arrastrarlo en dirección a la oveja que quiera guiar hacia el corral, en el momento en que el perro entre en contacto con la oveja, esta dejará de huir para pasar a seguir las instrucciones del perro, que serán ir en la dirección en que este la empuje.

[30]Raza de perro propia de Cataluña. Es un perro pastor que se utilizaba sobre todo para vigilar los rebaños durante el día a la zona de los Pirineos y a la zona ocupada por la trashumancia relacionada con esta zona.

4 Lamb Pastor multijugador

Para realizar la parte multijugador de Lamb Pastor se decidió optar por que los dos jugadores interactuaran conjuntamente en la misma escena, de manera que el nivel de competitividad fuese alto.

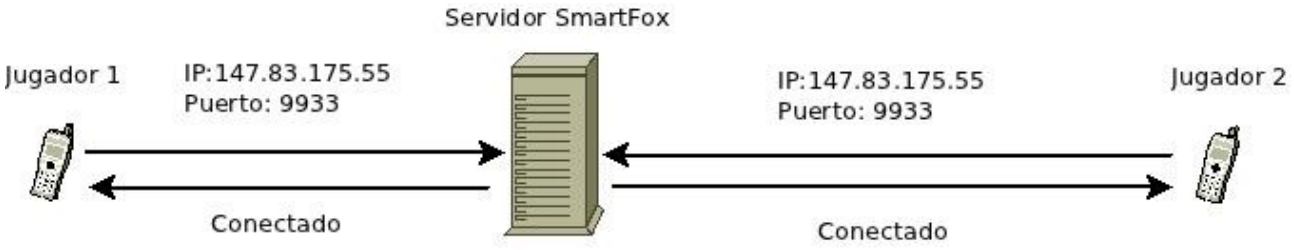
El mecanismo es igual que el de un jugado, es decir, el número de ovejas se mantiene y el tiempo es el mismo, pero en esta ocasión hay dos perros en la escena.

4.1 Conexión con el servidor

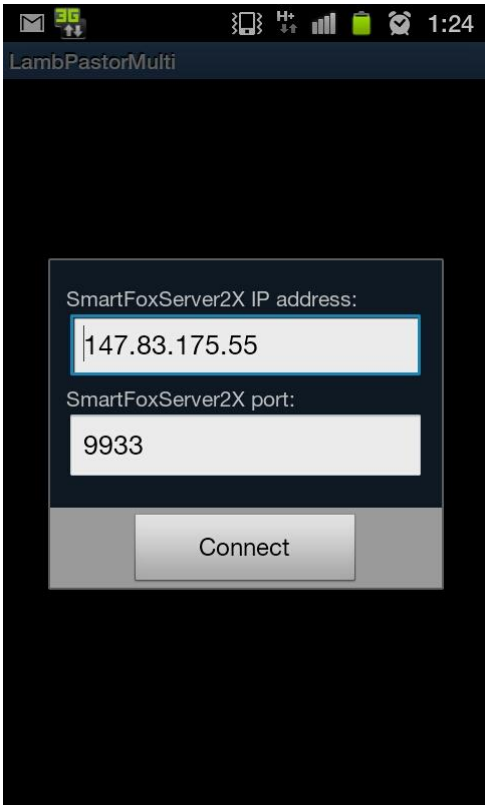
A continuación pasaremos a detallar como funcionan la conexión con el servidor y la negociación de los datos.

4.1.1 Realización de la conexión

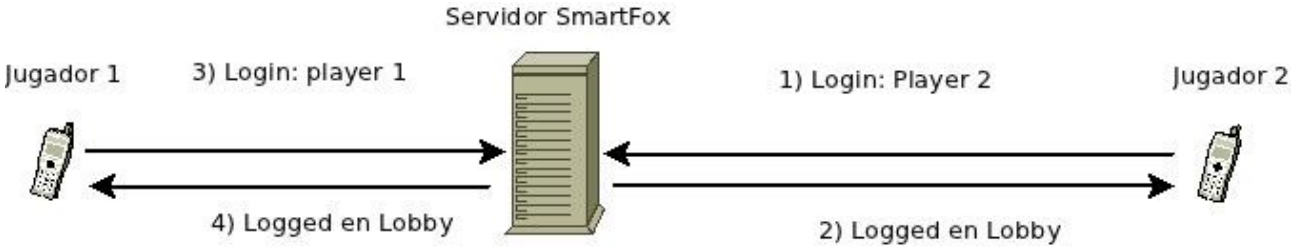
Para realizar la primera conexión con el servidor debemos indicar su dirección IP, y al puerto que queremos conectar (por defecto el 9933).



En la aplicación la pantalla de conexión tendrá el siguiente aspecto:

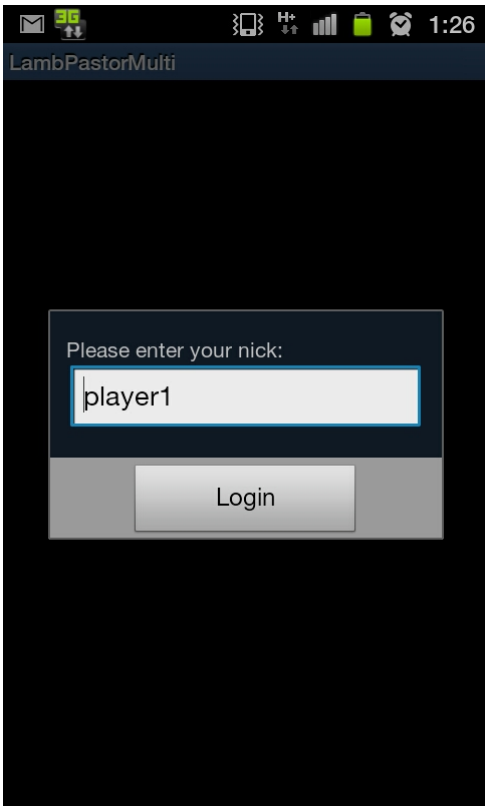


Una vez la conexión con el servidor se ha realizado es hora de loguearnos con un nombre en el servidor. Para ello escribiremos un nombre y se lo enviaremos al servidor, en el momento que el servidor nos dé el visto bueno accedemos a la sala principal *lobby*[31]. Una vez hecho esto esperaremos a que un usuario se conecte antes de crear la nueva sala.

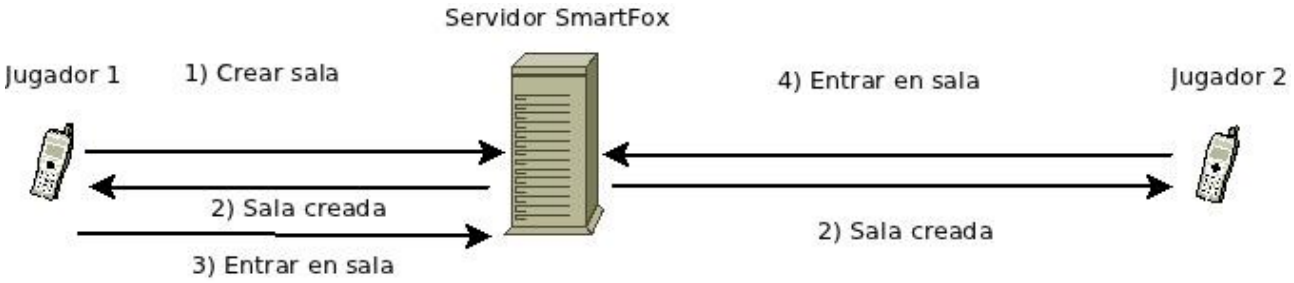


[31]Del ingles, Vestíbulo.

Interfaz de login en la aplicación:



Una vez detectamos que ya hay un nuevo usuario es el momento de crear la nueva sala para poder empezar la partida. El jugador envía la petición de crear una nueva sala al servidor, cuando esta esta creada, envía a los jugadores un aviso conforme dicha sala ha sido creado, y ambos jugadores entran para poder iniciar la partida.



Una vez los usuarios están dentro de la sala, inician la inicialización de la aplicación (carga de texturas, sonidos, físicas, etc.). Cuando ambos están listos mandan la señal al servidor conforme están preparados para iniciar el partido.

Una vez el servidor recibe la confirmación de que los jugadores están listos inicializa las posiciones de las ovejas (posiciones y caminos a seguir), y la posición inicial de los perros. Cuando el servidor ha generado toda esta información la envía a los jugadores de manera que ya pueden iniciar el juego.

Cuando finaliza el juego podemos tener 3 situaciones:

1. Un jugador abandone la sala: en este caso se mostrará un mensaje conforme el jugador se ha desconectado, y la aplicación vuelve al lobby.

2. Todas las ovejas son introducidas en el cerco: si esto ocurre el servidor comprueba quien ha metido más ovejas, y noticia a cada jugador el resultado.

3. El tiempo se termina: se seguirá el mismo procedimiento del punto 2.

4.2 Interacción durante el juego

Para poder realizar correctamente la sincronización entre los dos juegos se ha buscado reducir el problema lo máximo posible. Por ello se decidió que los clientes enviarán información relativa a los cambios que se producían en su pantalla de manera que el flujo de información sería reducido y de esta manera el tráfico de red se volvería pequeño.

Por lo tanto un usuario enviará información cuando:

- El usuario mueva al perro.
- El usuario mueva a una oveja.

4.3 Definición de las nuevas clases de LambPastor

Ha sido necesaria la adaptación de la aplicación android que maneja el juego en solitario para poder manejar todos y cada uno de los eventos que genera el servidor SmartFoxServer, para esto hemos tenido que añadir una serie de métodos al código fuente de la aplicación android, los cuales pasamos a detallar a continuación.

LambPastorMultiActivity

La función de esta clase es crear una pasarela entre la aplicación local y el servidor, realizando una conexión. A continuación veremos los diferentes elementos que se lanzan dentro de esta aplicación

- Variables definidas

- userName: variable que contiene el nombre del usuario.
- SfsClient: esta variable es la encargada de obtener información acerca del servidor (tales como número de usuario, salas disponibles, etc.).
- SFS_ZONE: esta variable contiene la zona a la que se va a conectar nuestro cliente, en este caso *"LambPastorExtension"*.
- GAME_ROOMS_GROUP_NAME: contiene el nombre de grupo de las partidas donde crearemos la partida en este caso lo hemos llamado *"games"*.
- EXTENSION_ID: nombre de la extensión.
- EXTENSIONS_CLASS: nombre de la clase de la extensión en el servidor, en este caso *"sf2xLambPastor.LambExtension"*

- *Método onCreate*

Método por defecto de android que lanza la aplicación, dentro de este método además se define *layout*[1] que va a usar la aplicación, y se llama a los métodos *initSmartfox()* y *resetUI()*.

- *resetUI ()*

Esté método quita de la interfaz todos los *dialogs*[2] que se encuentre en ella.

- *createRoom ()*

Este método se encarga de crear la sala de juego en el servidor. Configura los ajustes de la sala (nombre del grupo, nombre de la sala, máximo número de jugadores en la sala, indicar que es un juego, número máximo de espectadores, a que extensión va asociada y a que clase de extensión va asociado).

- *InitGame ()*

Este método inicia el juego una vez el jugador ya ha accedido a la sala de juego.

- *getConnectDialog ()*

Este método se encarga de lanzar el *dialog* que nos permitirá realizar la conexión con el servidor. El *dialog* está compuesto por dos cajas de texto (una para la IP, y otra para el puerto) y un botón que es el que recoge la información de las cajas de texto y realiza la conexión con el servidor.

- *getLoginDialog ()*

Este método lanza el *dialog* de login en el servidor. Este *dialog* está compuesto por una caja de texto que será donde escribiremos nuestro alias, y un botón que será el encargado de enviar la petición de login con ese nombre al servidor.

- *initSmartFox ()*

Este método se encarga de inicializar la variable *sfsClient* además de registrar en el cliente los siguientes eventos:

- CONNECTION: será atendido cuando el usuario realice la conexión con el servidor.
- CONNECTION_LOST: será atendido cuando el usuario pierda la conexión con el servidor.
- LOGIN: será atendido cuando el usuario se loguee en el servidor.
- ROOM_JOIN: será atendido cuando un usuario acceda a una sala.
- USER_EXIT_ROOM: será atendido cuando el usuario salga de una sala.
- ROOM_ADD: será atendido cuando sea añadida una sala.
- ROOM_REMOVE: será atendido cuando una sala sea eliminada.
- ROOM_JOIN_ERROR: será atendido cuando haya un error en el proceso de login.

Estos eventos serán atendidos en el método *dispatch* que explicaremos más adelante.

- *onDestroy ()*

Esté método se encarga de quitar los eventos relacionados con el servidor ya que este método se llama cuando la aplicación va a finalizar.

- *connectToServer(final String ip, final int port)*

Método que es llamado desde el *dialog* de conexión con el servidor. Este método crea un nueva thread para realizar la conexión con el servidor sin que esto haga que la interfaz quede bloqueada.

- *dispatch (final BaseEvent event)*

En este método es donde se atiende todos los eventos que envía el servidor. A continuación explicaremos que se hace en cada uno de ellos:

- CONNECTION: quita el *dialog* de conexión, y lanza el de *login*.
- CONNECTION_LOST: vuelve a lanzar el *dialog* de conexión.
- LOGIN: se añade el usuario a la lista de usuarios.
- ROOM_JOIN: incrementa el número de usuarios de la sala en 1.
- USER_EXIT_ROOM: decrementa el numero de usuarios de la sala en 1.
- ROOM_ADD: se añade la sala a la lista de salas y el usuario que la ha creado entra en ella.
- ROOM_REMOVE: se elimina la sala de la lista de salas.
- ROOM_JOIN_ERROR: vuelve a lanzar el *dialog* de login.

- *updateGameList ()*

Actualiza la lista de salas.

LambGame

Esta clase es la que se encarga de arrancar el servidor y realizar todo el intercambio de datos entre la aplicación android y el servidor que aloja la extensión que controla nuestro juego online.

A continuación pasaremos a detallar los diferentes métodos que componen esta clase, así como los métodos que se han modificado del LambPastor original.

En este apartado sólo se analizarán los nuevos métodos que se han añadido/modificado a la clase de LambPastor (LambGame es la misma clase que LambPastor pero añadiendo la interacción con el servidor).

- Variables definidas

- gameStarted: variable que indica si el juego está iniciado.
- Sfs: esta variable es la encargada de obtener información acerca del servidor (tales como número de usuario, salas disponibles, etc.).
- Moves: variable de tipo enumeración para realizar la programación de cara al servidor más cómoda (contiene start, stop, move, win, tie).
- Winner: variable para determinar si eres el ganador de la partida.
- Loser: variable para determinar si eres el perdedor de la partida.
- Tie: variable para determinar si la partida ha acabado en empate.
- Player1: variable que determina si eres el jugador 1.
- miNombre: nombre del usuario.
- Player1Name: nombre del jugador 1.
- player2Name: nombre del jugador 2.

- *initGame*

Este método añade tres eventos para que sean atendidos por la aplicación:

- EXTENSION_RESPONSE
- CONNECTION_LOST
- USER_EXIT_ROOM

Una vez añadidos, el usuario le envía al servidor la confirmación de que está listo para para iniciar el juego. Una vez hecho esto lanzamos un bucle que espere a que la variable *gameStarted* esté a cierto.

- *Handler*

Un *handler* o manejador es una llamada a una subrutina de manera asíncrona.

El *handler* que hemos definido atiende los mensajes que nos manda el servidor para saber el resultado de la partida. Los mensajes que nos puede enviar el servidor son:

- winner: has ganado la partida.
- looser: has perdido la partida.
- tie: has empatado la partida.

- *dispatch*

En este método es donde se atiende todos los eventos que envía el servidor. A continuación explicaremos que se hace en cada uno de ellos:

- CONNECTION_LOST: vuelve a LambPastorMultiActivity y devuelve que ha habido una desconexión del servidor,
- EXTENSION_RESPONSE: en este apartado es donde se concentra la interacción con el servidor. El servidor puede mandarnos tres parámetros diferentes:
- start: cuando el servidor reciba las dos peticiones de los jugadores para iniciar la partida, el servidor invocará este método que llamará a *startGame ()*.
- Move: cada vez que el servidor reciba el movimiento de una oveja o un perro, el servidor invocará este método que llamará a *MoveReceived ()*.
- win/tie: cuando el juego termine el servidor invocará este método que llamará a *ShowWinner ()* que mostrará el usuario que ha ganado.
- USER_EXIT_ROOM: cuando el otro usuario salga de la sala se atenderá este evento que lanzará un *dialog* mostrando un mensaje de que el otro usuario se ha desconectado, y cuando pulsemos el botón nos enviará a *LambPastorMultiActivity*.

- *initOvejas ()*

Este método sustituye al que había anteriormente en LambPastor. Este método es llamado después de que el servidor haya inicializado las ovejas, por lo tanto cogemos la información que nos ha mandado e inicializamos las ovejas con los valores que nos ha pasado el servidor.

- *showWinner ()*

Este método hace saber al *handler* si somos el ganador, perdedor o la partida ha resultado en empate.

- *moveReceived()*

Una vez el servidor nos envía la información de los movimientos realizados. Primero comprobamos si lo que han enviado es información acerca de las ovejas, en caso afirmativo obtenemos que juga-

dor lo ha enviado en caso de ser nosotros mismos obviamos el mensaje, y si ha sido el otro usuario, actualizamos nuestras ovejas.

En caso de no ser el movimiento de una oveja, es que se ha enviado el movimiento de un perro, como en el caso de las ovejas comprobamos que no hayamos sido nosotros los que hemos enviado la información y la actualizamos, encaso contrario obviamos la información.

- *startGame ()*

El servidor envía los datos de los jugadores, de los perros, y de las ovejas. Respecto a los usuarios el servidor envía el nombre de cada uno de ellos, en cuanto a los perros envía la posición en x e y, y por último envía la posición de cada oveja y el camino que van a seguir.

Para finalizar éste método pone la variable *gameStarted* adquiere el valor cierto.

- *gameOver ()*

Desactiva todos los eventos que se habían registrado, el usuario accede a la sala principal *(lobby)* y por último vuelve a *LambPastorMultiActivity*, y devuelve que se ha salido de la sala de forma normal (sin que haya salido ningún usuario de forma inesperada).

- *Envío del movimiento de los perros y las ovejas*

Cuando hacemos que el perro se mueva arrastrándolo por la pantalla, enviará al servidor todas las posiciones de la pantalla por las que pasa.

En el caso que el perro colisione con una o más ovejas, enviará al servidor tanto su posición como de las ovejas con las que ha colisionado.

5 Análisis de Costes

En este apartado pasamos a detallar el coste total aproximado que supondría llevar a cabo el proyecto planteado en este documento. Con el fin de hacer más sencillo la forma en que se plantean los costes derivados, los expondremos en una tabla.

En la tabla incluiremos tanto los costes derivados de los RR.HH. asignados para llevar a cabo todas las tareas de desarrollo y configuración necesarias para poner el sistema apunto, como los costes derivados del hardware necesario para soportar la implementación planteada.

Hardware	Precio Estimado
Ordenador(cumpliendo requisitos mínimos)	600€
Hosting Server	1000€
Dispositivo Android(cumpliendo requisitos mínimos)	150€

Software	Versión	Licencia	Precio Estimado
Ubuntu	11.04	GPL	Gratuito
Eclipse	Indigo 3.7.1	GPL	Gratuito
Android SDK	R16	GPL	Gratuito
Eclipse Plugin ADT	15.0.1.v201111031820-219398	GPL	Gratuito
SmartFoxServer		GPL	Gratuito
LibreOffice	3.3.4	LGPL	Gratuito
ArgoUML	0.34	GNU GPL	Gratuito

RR. HH.	Horas estimadas	Precio/Hora €	Precio Estimado
Diseñador (Adaptación Android)	60	60	3600€
Diseñador (Server)	90	60	5400€
Programador (Adaptación Android)	100	60	6000€
Programador (Server)	200	55	10500€

Concepto	Precio Estimado
Hardware	1750€
Software	0€
RR.HH.	25500€
Coste Total: 27250€	

6 Dificultades Encontradas

Durante el desarrollo de este proyecto las mayores dificultades con las que nos hemos encontrado ha sido la integración de las dos API's externas para llevar a cabo los diferentes desarrollos de los que se compone este proyecto.

La mayor dificultad que ha surgido ha sido el lanzamiento de 2 threads simultáneos por parte de la API AndEngine, en la cual se basa el desarrollo de la aplicación Android, con el fin de mantener el juego actualizado en todo momento.

El motivo de estos 2 threads, es poder manejar la rotación del dispositivo de manera rápida sin afectar al usuario de forma significativa. El problema surgía al crear dos conexiones con el servidor que hacían que este expulsará al usuario por exceso de conexiones.

Otro de los problemas que mayor horas de número de trabajo representó, fue encontrar la manera apropiada de gestionar y manejar todos y cada uno de los eventos que genera el servidor, para poder mantener actualizada en todo momento la interfaz de usuario.

Era necesario tener en cuenta la limitación que supone establecer este tipo de conexiones desde un dispositivo móvil, ya que imperaba la necesidad de reducir al máximo el tráfico enviado.

7 Conclusiones

Los objetivos definidos en la fase inicial del proyecto han sido asumidos con éxito en todos los casos. Se ha llevado a cabo un estudio profundo de la estructura de conexiones de datos inalámbrica ofrecida por el sistema operativo android, albergando así muchos conocimientos sobre las API's que nos ofrece el sistema.

El principal objetivo que se debía asumir durante el proyecto era establecer una conexión entre dispositivos Android, de forma inalámbrica, mediante el uso de un servidor alojado en un host público. Para asumir este objetivo hemos estudiado tanto diversos servidores de comunicaciones como las comunicaciones de android, llegando a tener un profundo conocimiento de partes hasta ahora desconocidas sobre este, y asumiendo grandes conocimientos sobre el manejo de eventos y conexiones de datos en servidores de comunicaciones.

Un objetivo adicional era el manejo de eventos e intercambio de datos entre dispositivos, estudiando las posibilidades del servidor escogido, para lo que se ha llevado a cabo un estudio de los ejemplos y java-docs proporcionados con este, con la finalidad de entender su funcionamiento y ponerlo en práctica, en la aplicación.

Durante todos los estudios realizados tanto del servidor SmartFoxServer, como de las API's de android, hemos obtenido una grata experiencia, viendo como los conocimientos iniciales sobre la materia se han visto claramente beneficiados, proporcionando una visión mucho más amplia de todas las posibilidades que nos ofrecen ambas plataformas.

8 Bibliografía

[1] Beginning Android 3. Murphy, Mark. Editorial Apress – 2011.

[2] Pro Android 3. Komatineni, Satya - MacLean, Dave – Hashimi, Sayed . Editorial Apress – 2011.

[3] Pro Android Games. Silva, Vladimir. Editorial Apress - 2010.

[4] Android Official Developer Program: [Consulta: 15/10/2011]
http://android.developer.com

[5] SmartFoxServer2X Support: [Consulta: 22/09/2011]
http://www.smartfoxserver.com/forums/

[6] Foro especializado en Android: [Consulta: 30/09/2011]
http://www.android-spa.com

[7] Foro especializado en Android: [Consulta: 21/10/2011]
http://www.stackoverflow.com

[8] Wikipedia: [Consulta: 14/11/2011]
http://www.wikipedia.org

[9] Java API: [Consulta: 18/11/2011]
http://docs.oracle.com/javase/1.5.0/docs/api/

9 Anexos

9.1. Anexos 1 – Diagrama de Clases